Dataspaces: Where Structure and Schema Meet

Maurizio Atzori[*] and Nicoletta Dessì

Abstract. In this chapter we investigate the crucial problem that poses the bases to the concept of dataspaces: the need for human interaction/intervention in the process of organizing (getting the structure of) unstructured data. We survey the existing techniques behind dataspaces to overcome that need, exploring the structure of a dataspace along three dimensions: *dataspace profiling*, *querying and searching* and *application domain*. We will further explore existing projects focusing on dataspaces, induction of data structure from documents, and data models where data schema and documents structure overlaps will be reviewed, such as Apache Hadoop, Cassandra on Amazon Dynamo, Google BigTable model and other DHT-based flexible data structures, Google Fusion Tables, iMeMex, U-DID, WebTables and Yahoo! SearchMonkey.

1 Introduction

Data integration has emerged over the last few years as a challenge to improving search in vast collections of structured data that yield heterogeneity at scale unseen before. Current information systems and IT infrastructures are mainly based on the exchange of strongly-structured data and on well-established standards (database, XML files and other known data formats). Nevertheless, enterprise and personal data handled everyday are mostly unstructured (estimates range from 80 to 95%), i.e., their contents do not follow any rigid schema or format (e.g., text files or bitmap images), therefore not

Maurizio Atzori · Nicoletta Dessì
University of Cagliari
e-mail: `atzori@unica.it,dessi@unica.it`

[*] The work of Dr. Atzori has been done within the project *Unstructured Data Integration for Dataspaces (U-DID)* founded by "RAS PO Sardegna FSE 2007-2013 L.R.7/2007".

allowing complex queries or strong integration within automatic enterprise business processes or workflows. Data in enterprise computer systems (email, report, web pages, customers and supplier records) [1], in social networks [2] and in personal computers (documents, images, videos, chats, short messages, favourite web pages, appointments) [3] are often unused, or even forgotten, because of the weak support this generation of IT infrastructures (mainly based on databases management systems) offer for non-structured data.

Improving data integration in such heterogeneous information spaces leads to a fundamental question: the presence of a plethora of structures limits or makes too difficult the definition of some kind of global structure? In other words, are traditional approaches for engineering data over selected resources (i.e., mediator architectures and XML-based solutions) the only possible way for data integration or can we extend the data structuring concepts to improve data integration in such contexts?

The research community has recently proposed the concept of *dataspace* [4, 5] as a new scenario for structuring information relevant to a particular organization, regardless of its format and location, and capturing a rich collection of relationships between them. The elements of a dataspace [5] are a set of participants (i.e., individual data sources) and a set of relations denoting the relation in which the participants are. In this sense, a dataspace is an abstraction of databases that does not require data to be structured (i.e., in tabular form), with a minimal "off-the-shelf" set of search functions based on keywords. The key idea is to enhance the quality of data integration and the semantic meaning of information without an *a priori* schema for the data sources [6, 7]. Advanced DBMS-like functions, queries and mappings are provided over time by different components, each defining relationships among data when required. Integrated views over a set of data sources are provided following the so-called *pay-as-you-go* principle (i.e., the more you give the more you get, in an incremental and continuous fashion) that is currently emerging on the Web [8, 9].

The dataspace concepts have been presented in a visionary way [10, 4, 5] and their implementation on global scale opens new research challenges. From the point of view of metadata, dataspaces may be seen as a generalization of both DBMS and SE (search engines), where the schema of the former meets the document structure of the latter. In fact, in the context of databases, a schema is the set of metadata, relations and constraints regarding data. In other words, it is everything that is stored but the data itself. In search engines, the focus is instead on the documents, that are usually semi-structured (XHML), with ranking algorithms exploiting metadata such as links, headers and title. Despite this, the user cannot perform search using the structure of the document; they are just a set of words. Dataspaces aim at having the benefits of both DBMS and SE with minimal efforts for end users.

In this chapter we investigate the crucial problem that poses the bases to the concept of dataspaces: the need for human interaction/intervention in the process of organizing (getting the structure of) unstructured data. We survey the existing techniques behind dataspaces to overcome that need, by using algorithms for structure induction and flexible data models that take into account the diversity among data sources (e.g., text documents, pictures, tables) while leveraging all the knowledge about each document structure.

Our survey will explore the structure of a dataspace along three dimensions:

Dataspace profiling. Analogous to databases profiling [11], this dimension will analyze how recent proposals differ in defining the internal structure (components and relationship) of a dataspace [12, 13].

Querying and searching. Querying and Searching represent one of the main services supported by a dataspace [6, 7, 14]. Our review will investigate how differences in dataspace profiling will affect the formulation of queries on top of all participants in dataspaces.

Application domain. A dataspace defines a global "virtual location" connecting data from diverse application domains [4, 15]. This dimension will explore the characteristics of the new types of functionalities the dataspace enables and the new possibilities it opens within domain-specific applications.

Existing projects focusing on dataspaces, induction of data structure from documents, and data models where data schema and documents structure overlaps will be reviewed, such as Apache Hadoop [16, 17], Cassandra [18] on Amazon Dynamo [19], Google BigTable model and other DHT-based flexible data structures [20, 21], Google Fusion Tables [22], iMeMex [3, 23], U-DID, WebTables [24] and Yahoo! SearchMonkey.

Chapter Organization. The chapter is organized as follows: in Section 2 an introduction to data structuring is given; Section 3 gives a review of the approaches in data integration; Section 4 presents dataspaces, a new concept born from the data integration community, further analyzed over different dimensions in Section 5; finally, Section 6 describes existing and ongoing projects in the field of massive structured data management with no predetermined schema given.

2 Data Structuring

The definition of a data structure allows for organizing, storing querying data in a software system so that it can be managed efficiently. Besides technical aspects related with data to be stored, a data structure expresses an organization of logical concepts of data and its careful choice often allows the most efficient algorithms to be used. As well, being some data structures highly specialized to well-defined tasks, the choice of a specific data structure depends on the kind of application. For instance, hash tables are particularly suited for compilers, while queries in very large databases are usually made more effective by using B-trees. The implementation of a data structure usually requires a schema describing how the instances of that structure can be stored, accessed and manipulated by applications. Thus a schema addresses both technical features related with the management of data (i.e., field formats and data types) and some aspects concerning the contents and the meaning of the data, such as the cardinality integrity, the referential constraints, etc. In some sense we may argue that the schema itself is a structure of metadata expressing semantic properties and exhibiting varying capabilities and expressiveness in supporting declarative access to and manipulation of data. This observation motivates the assertion that the efficiency of a schema cannot be analyzed separately from the operations that may be performed on its instances and the properties of those operations, including their efficiency and their cost.

To investigate more comprehensively the effectiveness of a schema, let us present the possible ways of structuring data and discuss the effectiveness of the related managing procedure. Regarding their structure, data can be classified into the following four groups.

Structured data is organized according to a schema in order to allow retrieval of data via a structured query language. The schema is defined in terms of constructs of some data model, for example, the relational model or the object oriented model. Data is formatted according to the schema prior to populate the database. The very front end of structured data deals with modifying the schema to make it more suitable as requirements for new types of entities and relationships arise. For example, in business environment, changes to the existing schema are rare and avoided wherever possible with new linked tables being created rather than exiting table modified.

Unstructured data includes word processing documents, pictures, digital audio and video, emails and PDF attachments, files and folders, PDFs as blobs. The conventional hypertext-based Web provides the illusion that the relationship between two linked documents is defined by some schema when in fact it is implicitly expressed by typed links in HTML.

Semi-structured data is generally regarded as data that is self-describing, i.e., the schema may not be known in advance but schema information may accompany the data, e.g., in the form of XML tags or RDF statements. In the context of the Semantic Web, researchers have recently sought to provide facilities for semantic annotation which assigns to an annotation a reference to a concept within an ontology rather then an entity type [25]. As well, data stored in a text form is considered semi-structured data in that it may contain a few structured fields (such as title, sections, authors etc.) that, usually, are not filled in. Without knowing the document content, it is difficult to formulate effective queries for analyzing and extracting useful information from data. Queries that contain keywords are the simplest search method on the Web whose growth results in impressive amounts of information potentially relevant to the user, but very disorganized at the moment. Usually, query results contain a large amount of redundant information and it is difficult to estimate relevant values from whole content that grew exponentially with the growth of Internet. Precision and Recall [26] are two basic measures for assessing the quality of query answers.

Partially structured data where information consists partly of some unstructured data conforming to a schema and partly as free text [27]. It is generally regarded as data that is self-describing: in semi-structured data there is not a schema defined but the data itself contains some structural information, such as XML tags. In contrast, the *text* in partially structured data has no structure, while other parts of the document may be structured. Examples include accidents reports, a databases such as SWISS-PROT [28] that includes comment fields containing unstructured information related to structured data.

3 Data Integration: The Story so Far

The advent of the Web led to the participation of the users into the content creation and application development process lowering the barrier to publish and accessing information. Data and information management is becoming increasingly complex as more computational resources are made available and more data is produced as part of large scale collaborative activities. This results in a rapid increasing of loosely structured heterogeneous collections of data and documents coming from a variety of information sources, and leads to two fundamental and related questions:

1. Can we define and/or discover latent structural characteristics and regularities on data coming from a constantly growing number of heterogeneous information sources?
2. How traditional techniques for information retrieval and data mining can be adapted with uniform capabilities in order to be effective over this web-scale heterogeneity?

The above questions have received considerable attention within the research community. The resulting proposals provide various benefits and articulate data integration according to the following approaches.

3.1 Schema Mapping

Schema mapping combines data residing in different sources under a single integrated global view that provides a uniform query interface by transforming the original query into specialized queries over the respective data sources. This means to construct data elements or a mediated schema between different data models to meet the requirements of end users. A well-defined set of queries can be formulated, each one having a precise answer. Since the source schemas are independently developed, they often have different structure and terminology. Thus, a first step in integrating the schemas is to identify and characterize these inter-schema relationships. This is the schema matching process. It is performed to find relationships between concepts in each schema, i.e., finding the semantic correspondences between elements of two schemas. The input information provides element names, data types, description, constraints and so on. The instance data is exploited to characterize the content and semantics of schema elements. Then the matching elements can be unified, resulting in a set of mapping items. Comprehensive surveys of schema matching approaches are presented in [29, 30, 31].

Before any service or application can be provided, a mapping schema must be defined, aware of the precise relationships between the terms used in each schema or data source. Schema matching nowadays is performed manually by domain experts. Obviously, it is a tedious, time consuming, error prone and expensive process. In general, it is not possible to fully automatically determine all correspondences between two schemas due to their semantic heterogeneity. Moreover autonomous changes in the sources require to modify the matching schema. In recent years, significant progresses in srning, database and data mining allow for partially automating schema matching. However, this approach is poorly suited in application environments (such as scientific communities, personal computing, virtual organizations, etc.) where it is difficult to capture consensus on a single schema.

3.2 Keyword-Driven Queries

Data integration appears with increasing frequency in a variety of situations and is critical when the volume and the need to share existing data tremendously increase according to the Web's unconstrained growth. It has been pointed out [10] that "traditional data integration techniques are no longer valid in the face of such heterogeneity and scale" and new proposal are needed. Schema matching solution is less effective for Web search engines and for on-the-fly data integration whose application scenario makes difficult and often impossible to get the right mappings. In this case, it is rather preferable an approach that does not require a matching process over the data source and allows the user to pose keywords query or to define structural search constraints. The query model is either based on keywords over all available sources (bag-of-words query model) or on defining XPaths expressions over a XML schema (XML query model). It is worth noting that the bag-of-words query model does not perform any data integration while the choice of a specific XML Schema can seriously impact the effectiveness of a query.

3.3 The Web of Data

Recent research aims to overcome some of the problems encountered in data integration by addressing not the definition of the best integrating structure, but instead the solution of semantic conflicts between heterogeneous data sources. In semantic web settings, a popular strategy to solve this problem, the approach involves the use of ontologies whose role is analogous to the mapping schema. Ontologies express semantic relationships over data in order to resolve semantic conflicts. As of 2006 the trend in semantic integration has favored integrating independently-developed ontologies into a single ontology according to the vision of a Web of Data connecting data from diverse domains with database-like functionality. The aim is replacing a global information space of linked documents to one where both documents and data are linked [32]. The following rules, also denoted as *Linked Data principles* [13], provide directions for publishing and interlinking data on the Web in a way that all published data become part of a single Web of Data:

1. Use the URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things.

Started in 2007 by the W3C Linking Open Data Community, the Web of Linked Data is a public repository consisting of hundred of datasets published by heterogeneous organizations (Universities, Companies, Governments) and comprising data of diverse nature such as music, films, scientific publications, genes, statistical data, television programs etc. The Linked Data approach allows data to be easily discovered and used by various applications. According to a navigation-based query model, RDF links allow client applications to navigate between data sources and to discover additional data. However few mechanisms exist for discovering relevant sources and getting the data automatically.

4 Dataspaces

By expressing a query to a web search engine users pursue a variety of concerns. They interact opportunistically with the web environment, wondering what the components they encounter can contain and then exploring the consequence of various possible browsing actions. They search for information they imagine might exist, no matter how it is structured. They seek explanation for events and relationships they may note. Much of the interaction with the web environment results in carrying on browsing activity on linked sites and searching by keywords. The problem with this kind of approach is that it generates unmanageably huge datasets with no internal structure, no principle for integrating information since users cannot query data using a structured query. A recent proposal within database community is called *dataspaces* [4, 5] that try to offer uniform access to heterogeneous data sources. The key idea of a dataspace is to offer best-effort queries that return possibly related data. In contrast with the data integration approaches presented previously, a dataspace does not consider the existence of a matching schema nor assumes that complete semantic mappings have been specified. It raises the abstraction level at which data is managed in order to manage all information relevant to a particular organization regardless of its format and location, and relationships between them.

A dataspace consists of components (also called participants) and a set of relationships among them. The participants are individual data sources such as Relational Databases, XML schemas, unstructured or partially structured information. Each participant knows the relationships to other participants and their source, i.e., if the participant is added by a user or automatically generated. As well, each participant contains information about the kind of data it contains, the data allocation, the storage format and which querying mechanisms are allowed. In detail, a dataspace considers two models of queries [6]: predicate queries and neighborhood queries. A *predicate query* is formulated according to a simple structure and allow users to specify keyword-based query. An example of predicate query is as follows: "a Professor named `Smith`, teaching `Database course`, at `University X`".

A *neighborhood query* is also a keyword-based query, but it explores all possible associations between data items. For example, a neighborhood query searching for "`Smith`" will return all courses Smith teaches, all Smith publications, people working with Smith etc. Broadly speaking, data resides in distributed systems, but there is not a single schema to which all the data conform to.

A dataspace management system provides keyword search over all data sources: when the user formulates a more sophisticated query, tighter integrations are created only if their benefits balance the effort for their definition. The core idea of a dataspace is to start just with a simple data structure to formulate predicate queries over all its data source and to produce additional effort to semantically integrate those resources only when it is absolutely needed. This means that a dataspace addresses the data integration problem in a *pay-as-you-go* fashion by postponing difficult and expensive aspects related to the schema matching process. *The schema is defined incrementally by discovering the latent structure of data.* It is only available when the query requires a structure to be expressed over heterogeneous datasets. Therefore we believe dataspaces may be intended as an abstraction where schema meets structure.

5 Dataspace Dimensions

As described in the previous section, the goal of a dataspace is to discover and verify latent structural properties among its datasets in order to provide users with a query interface. Indeed, a measure of how well structural properties are defined is the extent to which the resulting user interface is effective in querying the dataspace. A dataspace is neither a data integration system nor a user interface. It merely provides the best effort to call attention to the aspects of a query that may still require work for structuring data. There is no guarantee that there will be a structured answer to the query but discovering latent structural properties of data explicitly helps to get out the relevant information.

Database dimensions are explored in [12] where the proposed framework considers the life cycle of a dataspace as composed by a set of phases. Just like any traditional data integration software, a dataspace is initialized, deployed, maintained and updated. The paper details each phase with a view of eliciting dimensions over which existing dataspace proposals have varied.

Here we propose a completely different approach that considers dataspaces dimensions as defined by three fundamental concerns that can affect their effectiveness: dataspace profiling, querying and searching, application domain.

5.1 Dataspace Profiling

Similar to database profiling [11], dataspace profiling concerns with defining a set of components and relationships among them in order to offer a rich and dynamic context for user queries. However, it differs from data integration approaches in taking the broader-scope perspective on user query activity. There is not a mediated schema to which all the data conform to and data is hosted in a plethora of systems. Conversely, a dataspace is a concrete representation supporting the query processes a user might engage in while pursuing a particular concern. As such, this representation is necessarily incomplete and there is an infinity of possible representations that depend on the relationships among components. The best effort in profiling a dataspace aims at having a representation that provides good coverage of the possible query activity. An approach to building a dataspace would start with a definition of the kinds of components in the system. The nature of relationships between components is structural and sacrifices much of their semantics. Relationships could be used to organize components, but also to generate other components. Given a set of components $S = (X, Y, Z)$, possible relationships are as follows:

- X was manually created from Y and Z
- X and Z came from the same source at the same time
- X is a view or a replica of Y
- X and Y are created independently, but reflect the same physical system

The dataspace contains a detailed description of basic and semantic information about the data in the components. Specific applications automatically create relationships between the participants, and improve and maintain already existing relationships. Location and data mining techniques are used to discover relationships, but human attention is required for creation of relationships. Often, additional data management features (backup, recovery, and replication) are provided for components that have no or only limited data management functions. As well, cooperation among participants is enhanced by special support such as translation tables for coded values, classification and ratings of document, etc. Of course, relationships might fail in organizing even the majority of available components, but still it is useful both for providing structural insight about components and for generating new components. In terms of traditional database properties, a dataspace exhibits often common form of inconstancy about data (where the values come from and how they are created) that increases uncertainty. As stronger guarantees are required, it is necessary to develop formalisms to represent and reason about external lineage and promote agreements among the various owners of data sources. Among a few papers featuring dataspace profiling we mention [7] that underlines the importance of analyzing the structures and the properties exposed by an information source and investigates techniques for property and path analysis over a variety of data sources. Specifically, the paper presents *Quarry*, a software environment to browse and refine harvested metadata in scientific domains.

Originally conceived to provide browsing and querying services over a set of triples, Quarry accepts resource-properties-value triples (initially provided through scripts created by domain experts such as scientists) and automatically recognizes signatures, i.e., common property patterns for resources. Being each signature materialized in a multi-column table, a user can inspect a set of property-value conditions asserted on a collection of resources. Inspection is supported by an API that returns a set of unique properties used to describe a restricted set of resources and/or a set of unique values for a given property over a restricted set of resources. This allows the set of resources to be described by a path expression where each node of is a conjunctive query of propertyvalue pairs. From a merely technical viewpoint, a path type is a simple sequence of property names such as `belongs_to.is_author_of.has_title`. Profiling a dataspace means to explore the characteristics of these paths. While in a relational setting *following a path* generally means to join relations, in a dataspace *examining path characteristics* may mean, for instance, to explore whether instances of a specific type exist for every entity in a collection of participants, to calculate how many instances of a fixed type originate from participants, etc. Quarry follows a *pay-as-you-go* approach in discovering paths and aims at supporting users who don't have any technical knowledge about databases and, in general, about how the data is stored or managed.

5.2 Querying and Searching

A dataspace aims at supporting incremental refinement of automatic mapping using information from different resources with minimal effort. From the users perspective, the distinction between search and query disappears and the user should be able to iteratively refine and modify the previous query as well to formulate queries about the source of data that can be itself a new answer other than traditional answers. The focus is on finding methods to interpret queries in various languages on participants that support multiple data models and/or unstructured and heterogeneous. Due to data inconstancy, metrics must be defined for comparing the quality of answers and the efficiency of query processing techniques. To correct errors, results should be inspected by humans, but it is impossible at web scale. Hence, it is necessary to provide feedback from user on the quality of results. Popular approaches to get implicit feedback are to record which of the answers the user clicks on, to supply sample answer that would meet user needs, to observe which sources are viewed in the same session, etc. Proposed methods aim at managing the transitions among keyword querying, browsing and structured querying.

Following an information retrieval-based approach, [6] considers indexing support for predicate and neighborhood queries on heterogeneous data that are not semantically integrated. The basic idea is to build an index whose leaves are references to data items in the individual sources. Data is modelled by a set of triples either of the form $(instance, attribute, value)$ or of the form $(instance, association, instance)$. An instance is described by a set of attributes and linked to another instance by associations. Both instance and associations are extracted out of the data sources using a variety of methods. A query is formulated as a predicate that contains an attribute and some keywords $[k_1, k_2, \ldots, k_n]$. It is supported by an inverted list stored in a matrix where the cell at the i-th row and the j-th column memorizes the occurrence of the keyword k_i in the instance j. Synonyms and hierarchies are used to accommodate heterogeneity.

A pay-as-you-go method is proposed in [23]. Data is represented using a graph model and a technique is proposed to gradually model relationships in a dataspace through *trails* that include traditional semantic attribute mapping as well as traditional keyword expansions as special cases. A trail asserts a (bidirectional) correspondence between two keywords k_1 and k_2 meaning that the query Q_1 (k_1) induces the query Q_2 (k_2), and vice versa. By mean of trails, it is possible to define equivalences between any two sets of elements in the dataspace. A query is modeled by a graph and the query process consists of three major phases. First, matching detects whether a trail should be applied to a given query by checking the query with the left side of the itrail. Then, the right side of the trail is used to transform the original query graph. Finally, the transformed query is merged with the information provided by itrail definition to obtain a new query that extend the semantic of the original one.

Leveraging ideas from keywords search in a databases, work in [33] proposes a method for automatic adding new data sources and relating them to the existing ones. Given a set of databases that contain known cross-references, the user specifies a keyword-based query that is dynamically expanded into a query graph by a system, called Q. From each graph,

Q generates a conjunctive SQL query and associates a cost expression. Finally, Q provides a ranked view consisting of a union of conjunctive queries over different combination of the sources. This view is materialized and refined through user feedback. When the user registers a new source, i.e., a new database, the new source relevance is evaluated to the existing ranked views that are updated as appropriate if the relevance is found to be significant. The paper incorporates state-of-the-art methods from database [34] and machine learning literature [35] not only to automatically discover semantic links among data sources, but also to combine information resulting from multiple matching methods. Through experiments on actual bioinformatics schemas, authors demonstrate that their strategy is an effective step towards the ultimate goal of automating data integration. However, the consideration of heterogeneous and unstructured dataset, including Web sources, is an ongoing work.

5.3 Application Domain

One challenge of Web-based environments is to move from the idea of a local site towards networked collaborative environments that are requested to provide access to a variety of information and data sources. These environments are constantly in evolution and keep on increasing the aggregation and sharing of heterogeneous and geographically dispersed resources via temporary collaboration. In practice, this means that data sources are more or less invisible to the users whose search and query processes do not occur at a single location in a single context, but rather spans a multitude of situations and locations covering a significant number of heterogeneous data sources. Often, results are composed of parts that may themselves be data sources and no limit exists in this deep structure. For these reasons, dataspaces encompass more than just an integration data approach aiming to be an highly versatile and dynamic paradigm for integrating data on-demand. There are a number of domains where they can be defined to promote and mediate people's interactions with computers and other peoples. A basic step is to recognize that queries cluster around pervasive user search. Because users are allowed to influence the behavior of a dataspace by customizing their usage on his preferences, a dataspace implementation has to consider the context in which user operates. This context can spawn a staggering number of aspects, but the dataspace could restrict attention to those concerns that can be classified at similar level of abstraction. Often these concerns involve datasets whose heterogeneity is not perceived by the user who is, in turn, overwhelmed by the similarities of information. Structural differences akin to the various ways of performing a musical piece: people perceives the leitmotif but often would be enabled to appreciate technical differences in various executions. As such, we believe that the principles enabling a dataspace for a research environment [36] can substantially differ from those supporting access and manipulation of all of the information on a person's desktop with possible extension to personal information on the Web. To explain these differences next section presents a survey on existing projects in different domains.

6 A Roundup of Existing Projects on Managing Structured Data

In this section we analyze existing and ongoing projects and proposals focusing on dataspace-like approaches to data management or correlated topics such as induction of data structure from documents and data models mixing schema and document structure. When observing data management systems from a higher point of view than enterprise DBMSs, *structured storage* is the term generally used in literature [37, 18] referring to a large set of data management systems of which standard DBMSs are just a subset. Structured storage generalize DBMSs by not requiring fixed schemas and usually making distributed storage as a built-in feature in order to manage massive datasets transparently to the client. If we think of data handled as tables, structured storage data tend to grow horizontally by having weak constraints on the structure (columns) of the tuples. Apache Cassandra [18] (backed by Amazon Dynamo [38]), Google's BigTable [20, 21] and Apache HBase [17] are considered the most significant examples of structured storage, currently used in production environments from major web players. In the following we summarize the main aspects of each project, with a focus on the data model adopted to store or represent data and the query interface made available to users.

6.1 Google BigTable

BigTable [20, 21] is a data model and a structured storage system proposed by Google. It is built on top of Google File System (GFS for short) and Google Chubby locking service [39]. GFS [40] is a proprietary file system developed and used internally by Google to save data in commodity, cheap, with high failure-rate expected commodity servers. Data is usually appended to existing files, overwrites are rare. GFS is therefore distributed and like DBMSs it operates in the userspace (not being part of the underlying operating system kernel).

As in the Google paper of 2006 [21], BigTable is currently used by over 60 Google's products, including MapReduce, Google Reader, Google Maps, Google Book Search, "My Search History", Google Earth, Blogger.com, Google Code hosting service, Orkut, YouTube[1], and Gmail. According to the authors of the paper, it has been developed to enhance scalability, and better control performance characteristics, but in the conclusions they also

[1] Excluding the video storage.

state they "have gotten a substantial amount of flexibility from designing our own data model for BigTable". From the data model point of view, the main characteristic of this very influential work is that, despite standard relational databases, tables have not a fixed number of columns. They are sparse, distributed multi-dimensional sorted maps, sharing characteristics of both row-oriented and column-oriented databases. Thus, the data model behind BigTable can be summarized by the following function:

$$(row : string, column : string, time : int64) \rightarrow string$$

Collisions are not allowed (the corresponding item would be overwritten), and avoided by the model since the *time* key always change over time. The Application programming interface (API) offered to client applications is simple, while allowing very flexibility and full configurability. Rows are indexed by the *row* key, and are kept sorted by this key. Columns are also indexed, and kept grouped in a few families configurable in terms of access control, enabling compression, disk or memory storage, and data retention delays.

The usual operations performed against a BigTable are *get* and *put* to get/write an item with a specific key and *scan* to iterate over the items. Often the concept of BigTable is associated with a Google-patented approach to data analysis, called *MapReduce* [41, 42, 43]. Instead of accessing BigTable data with the basic API (get, put, scan), users exploit the functional programming approach of MapReduce, that forces the development of Divide-and-Conquer algorithms, therefore natively writing parallelizable data access.

After the publication of the paper, there has been a large number of attempts to implement the proprietary Google BigTable model, such as HyperTable, Open Neptune[2] and most notably the open source Apache's Cassandra and HBase (on top the Hadoop Core [16]), reviewed later in this section. Most of the code base of the Apache projects at hand has been donated and frequently contributed by web players like Facebook and Yahoo!, as described later on this section.

6.2 Apache Cassandra

Apache Cassandra is something similar to BigTable, initially developed and used by the Facebook team, and currently considered by the Apache Foundation a sort of open source implementation of the BigTable data model on top of a storage system similar to the DHT-featuring proprietary solution Amazon Dynamo [38]. A table in Cassandra is a distributed multidimensional map indexed by a key. The value is an object which is highly structured. Transactions are per-row, no matter the number of columns involved. Columns are grouped together into sets called column families, allowing personalization of column groups containing similar columns, as it happens in the BigTable

[2] See `http://openneptune.com/`

system. Cassandra also supports the use of multiple tables, although deployments often do not require this feature, expoiting instead the schema-free model of just one big table.

6.3 Apache Hadoop

Contributed by Yahoo! and used within their products, Hadoop is a Java framework on top of a number of supported file systems (including ad-hoc *Hadoop File System*, HDFS). As of 2008, Yahoo! Search Webmap was considered the largest in-production application featured by the Hadoop framework, with around 10K cores (approx 4K machines, also called nodes) powered by Linux. Facebook recently stated a 20 Petabyte Hadoop deployment for data warehouse of logs, being the largest known structured storage systems. Hadoop common framework exposes to clients a way to compute in parallel analysis over distributed data, in a MapReduce fashion [41, 42, 43] allowing, for instance, 1 terabyte of data sorted in order of 1 minute, assuming a reasonable amount of available nodes. Hadoop is an umbrella project for a number of subprojects; usually with the basic term of *Hadoop* it is meant the use of a framework to handle HDFS together to a job launcher/scheduler for MapReduce scripts. It is often distributed with some important subprojects, such as HBase, Hive, Pig and others. Interestingly, Hadoop has large support from open source community and big web companies, with further external projects such as *Ganglia* to monitor distributed Hadoop instances, and *Mahout* as a scalable machine learning library backed by Hadoop.

6.3.1 HBase

It is the component of Hadoop that, on top of HDFS, allows the BigTable data model in the Apache framework, with random read/write of the multidimensional associative array, leading to a column-oriented database management system [44, 17]. Efficient random access is not inherited by HDFS, but instead developed with appended-only log files, only occasionally merged with the remaining datafiles.

6.3.2 Hive

Hive [45, 46] is an extension of the Hadoop framework initially developed a few years ago by a Facebook team to data warehousing in a very scalable fashion, where a bunch of Mysql servers were failing in handling responsively lots of terabyte, with an average of 5Tb of data added every day. Hive adds to Hadoop the power of HQL, an SQL-like query language with a few extension w.r.t. Standard SQL. From a data model point of view it is interesting to note it handles sequences of data natively, by using commands `Expand` and `Collapse`. The former transforms the content of a column (actually a cell) into a sequence (i.e., another row with multiple columns), while the latter is the reverse operation, able to transform a multi-column row into a single value. The system also provides the command `Transform` to run MapReduce scripts within HQL.

As in standard SQL, the Hive data model consists of standard tables (i.e., typed columns, plus sequences and maps), and partitions (e.g., to partition tables by date) which are backed by directories in the HDFS subsystem. Bucketing can be applied to partitions (hash partitions within ranges, for sampling, optimizing hash joins, etc.). Regarding the implementation, tables and all the data of Hive are files in the HDFS (where directories determine the partitioning of tables), and the format of such files are arbitrary (e.g., comma separated, XML, YAML, JSON or any other chosen by the user). Sort of java "drivers" (usually very simple scripts, java classes called SerDe classes) must be provided in order to parse such unknown format files transforming them into Hive tables, although such parsers are already provided for most used formats. Another important component of the architecture is the *Metastore*, a component backed by any SQL-based management system as long as it provides a JDBC wrapper, such as MySQL or most notably Java Derby. Metastore is in charge of managing metadata, meaning for instance what in Oracle is contained in tables like `ALL_TAB_COLUMNS`; in the same way, Metastore saves metadata such as type of columns and any other information about the data which is not the data itself.

6.4 Apache CouchDB

Apache CouchDB is a document-based storage system characterized by its schema-free design, with a flat address space [47]. Documents can be considered semi-structured objects, essentially associative arrays where values can be either strings, dates, or even structured data like ordered lists or maps. The low level infrastructure allows data to be distributed among a number of server, with incremental replication and automatic bi-directional conflict detection between servers and user clients (in case of network failures or offline nodes). A free and commercial zero-configuration versions are currently supported by Cloudant [48].

6.5 DHT-Based Data Management Systems

The term Distributed Hash Table (DHT) is usually referred to systems able to handle very large hash tables over a medium-to-large set of nodes (machines). It is usually the ground level to construct higher level of abstraction on top of. One of the critical aspects of a DHT-based system is the routing protocol, that is, how to distribute (keys) over the network of nodes and how to search/recover them. Figure 1 shows a possible architecture featuring multi-level routing. Depending on the applications, needs of security

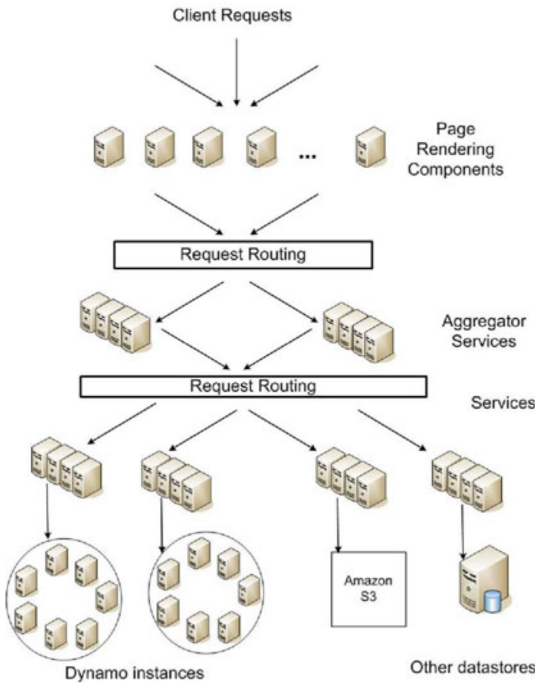

Fig. 1 An example of distributed architecture with clients (e.g., web browsers) facing front-end nodes, multi-level routing nodes and cloud-based storage servers (e.g., Amazon Dynamo instances and others)

(untrusted nodes, no single point of failure) or high availability (redundancy, decentralization), DHT systems can be much more complicated than a basic distributed hash table structure. They are currently used in several projects, including the projects presented so far and most peer-to-peer projects such as BitTorrent, eMule and Freenet [49]. From the data model point of view, at the lowest level they basically represent and implement associative arrays. Partitioning, replication, versioning, failure management and low latency vs. high throughput are the key aspects to be considered for each DHT-Based Data Management System [50].

6.6 Google Fusion Tables

Fusion Tables [22] is a web-based application featured by Google that allows table data to be uploaded into Google servers and then shared, queried in a structured way (including joins with other table sources) or used within other web applications. For example, supposing each row of a table contains latitude-longitude pairs, they can be showed through the Google Map service. This kind of data visualization and integration of structured data are easy to obtain given the tight collaboration among different Google applications.

It is an interesting proposal on a possible UI for data, although being most data in form of tables, it reduces the range of applications to structured-only data and appears similar to spreadsheets (with no computed values).

6.7 WebTables

Webtables is a shortcut for Tables in the Web. According to [24], almost 15 billion tables have been crawled by Google, many of them used for web site layout, but still a subset of 154M tables found by the authors as containing high-quality relational data. One of the contribution of the paper is that web-scale analysis of tables allowed the definition of an *attribute correlation statistics database* (called AcsDB) containing statistics about correlated attributes in the tables, useful in a number of contexts, including "semantic analysis" of table contents such as finding attribute synonyms and automatic join traversal of the tables.

6.8 Yahoo! SearchMonkey

Yahoo! SearchMonkey (SM) is a framework in charge of transforming the way search results are displayed. It is an interesting way to show how external structured data can be integrated to personalize web search results. In fact users like web site owners should share structured data (they may be in microformats[3], XML feeds, semantic web RDF or links to other structured data), then create a SearchMonkey application that handles the provided structured data to show a personalized search results. For instance, when someone queries Yahoo! for "best pizza in NY", if the search results include the web page personalized with SM, then that single result will appear differently, depending on the SM application behaviour (e.g., with review stars, a picture, a menu of site contents, etc.). In order to personalize "snipets" results we will use Yahoo! and user provided data input sources. Interestingly, data are structured into trees, where each data input is the root of a tree containing different type of information. For instance, trees are labeled `yahoo:index` or `com.yahoo.uf.hcard`, while nodes like `yahoo:index/dc:language` or `com.yahoo.uf.hcard/rel:Card/vcard:fn` respectively contain the language of the url at hand (e.g., "en") and the author of the page at that url (e.g., "Sam"). Recently Yahoo! marked this service as deprecated [51], and going to shift from a model where developers build lightweight apps to install on Yahoo! to one where publishers enhance their own site markup to produce similar results.

[3] For further information on microformats, see `http://www.microformats.org/`

6.9 iMeMex

iMeMex is the first attempt to develop an opensource dataspace for personal information management (PIM), run at ETH Zurich by a team leaded by Dr. Jens-Peter Dittrich. It follows the dataspace ideas in [4, 5], enabling the loose integration of different data sources (primarily files of any kind, emails, documents) through the use of user-defined views that give structure interpretation of the data belonging to each source. The iMeMex data model (called iDM) [3] is essentially a tree-based structure (with each node being any kind of object, e.g., a directory, a PDF file, a subsection of a LaTeX file), with links among nodes, leading therefore to a graph.

Despite the fact that different sources can be distributed over a network, the core architecture of the iMeMex system is essentially focused on a main centralized index (customizable and enhanced by the Apache Lucene full-text search engine), containing the information to answer user queries or to locate the required resource over the different data sources. This fact makes iMeMex less scalable w.r.t. projects described so far, but also more flexible on the structure and storage of data. It also has a rule-based query processor that is able to operate in three different querying modes: warehousing (only local indexes and replicas are queried), mediation (local indexes are ignored, queries are shipped to the data sources), and hybrid (combination of the former methods). Pay-as-you-go query rewriting techniques such as *iTrails* [23] (we described them in Section 5.2) are currently under development.

7 Conclusions and Future Work

In this Chapter we discussed the problems arising when large data with diverse structures need to be stored, managed, queried and integrated to improve user utility. Dataspaces are a promising framework to manage all kind of data in a uniform way, based on a simple data structure to formulate predicate queries over all its data source and to produce additional effort to semantically integrate those resources only when needed. The schema in the data is defined incrementally by providing the latent structure of data, thus letting the schema meet the structure. We reviewed the state of the art of both research results and commercial/opensource projects in the area.

Although very advanced, we believe these systems are still lacking of automatic algorithms to exploit unstructured data in order to learn their hidden structure. In fact, as we noted in the previous sections, most of the existing proposals are based on a pay-as-you-go user intervention, allowing keyword-only search over the data unless structure is explicitly given by the user. In an attempt to further analyze these aspects, *Unstructured Data Integration for Dataspaces* (U-DID for short), a recent ongoing project carried out by the authors of this chapter at the University of Cagliari, aims to investigate the topic of dataspaces as a possible kernel for data management in next-generation infrastructures. The project activities are focusing on defining a common model for managed data, uniforming the heterogeneity of data but meanwhile being able to represent the characteristics of each data type. The use of *data wrappers*, i.e., middleware algorithms able to convert "sequence of bytes" to structured data, either centralized (within a data management system) or distributed, will be fundamental for investigating the various possible dataspace architectures.

This important step involves the study of algorithms that extract a set of structured and integrable data (through metadata) out of input sequences with no previously-established format. Such algorithms can be divided into two groups: *fully unsupervised*, i.e., able to automatically compute the output without human intervention, and *semi-supervised*, i.e., taking into account online information (interactively provided) or offline information (datasets the algorithm will use to learn how to structure data). As we said, this kind of *structure learning* algorithms have not been investigated enough in literature and the project is therefore committed to contribute to this lack in the dataspace community.

Another approach under consideration is about the use of data mining techniques, well-studied for model extraction in the context of structured data. Clustering and association rules for structure extraction are definitely innovative, and seem to be promising, though they will require heavy adaptation of current algorithms.

References

1. Gounbark, L., Benhlima, L., Chiadmi, D.: Data integration system: toward a prototype. In: ACS/IEEE International Conference on Computer Systems and Applications, pp. 33–36 (2009)
2. Gatterbauer, W., Suciu, D.: Managing structured collections of community data. In: CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Online Proceedings, Asilomar (January 2011)
3. Dittrich, J.-P., Salles, M.A.V.: idm: A unified and versatile data model for personal dataspace management. In: Dayal, et al [52], pp. 367–378
4. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record 34(4), 27–33 (2005)
5. Halevy, A.Y., Franklin, M.J., Maier, D.: Principles of dataspace systems. In: Vansummeren, S. (ed.) PODS, pp. 1–9. ACM, New York (2006)
6. Dong, X., Halevy, A.Y.: Indexing dataspaces. In: Chan, C.Y., Ooi, B.C., Zhou, A. (eds.) SIGMOD Conference, pp. 43–54. ACM, New York (2007)
7. Howe, B., Maier, D., Rayner, N., Rucker, J.: Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In: ICDEW 2008: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering Workshop, pp. 270–277. IEEE Computer Society Press, Washington, DC, USA (2008)
8. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for dataspace systems. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 847–860. ACM, New York (2008)

9. Hedeler, C., et al.: Pay-as-you-go mapping selection in dataspaces. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD 2011. ACM Press, New York (to appear 2011)
10. Madhavan, J., Halevy, A.Y., Cohen, S., Dong, X.L., Jeffery, S.R., Ko, D., Yu, C.: Structured data meets the web: A few observations. IEEE Data Eng. Bull. 29(4), 19–26 (2006)
11. Marshall, B.: Data quality and data profiling - a glossary (2007), `http://www.w3.org/DesignIssues/LinkedData.html`
12. Hedeler, C., Belhajjame, K., Fernandes, A.A.A., Embury, S.M., Paton, N.W.: Dimensions of dataspaces. In: Sexton, A.P. (ed.) BNCOD 26. LNCS, vol. 5588, pp. 55–66. Springer, Heidelberg (2009)
13. Lee, B.: Linked data - design issues (2006), `http:/www.w3.org/DesignIssues/LinkedData.html`
14. Liu, J., Dong, X., Halevy, A.Y.: Answering structured queries on unstructured data. In: WebDB (2006)
15. Halevy, A.Y., Rajaraman, A., Ordille, J.J.: Data integration: The teenage years. In: Dayal, et al [52], pp. 9–16
16. White, T.: Hadoop: The Definitive Guide, 1st edn. O'Reilly Media, Sebastopol (2009)
17. Apache Foundation Software. Apache hbase, subproject of hadoop (2006), `http://hbase.apache.org/#Overview`
18. Lakshman, A., Malik, P.: Cassandra: a structured storage system on a p2p network. In: auf der Heide, F.M., Bender, M.A. (eds.) SPAA, p. 47. ACM, New York (2009)
19. Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 41(6), 205–220 (2007)
20. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. 26(2) (2008)
21. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: A distributed storage system for structured data (best paper award). In: OSDI [53], pp. 205–218
22. Gonzalez, H., Halevy, A.Y., Jensen, C.S., Langen, A., Madhavan, J., Shapley, R., Shen, W., Goldberg-Kidon, J.: Google fusion tables: web-centered data management and collaboration. In: Elmagarmid, Agrawal [54], pp. 1061–1066
23. Salles, M.A.V., Dittrich, J.-P., Karakashian, S.K., Girard, O.R., Blunschi, L.: itrails: Pay-as-you-go information integration in dataspaces. In: Koch, C., Gehrke, J., Garofalakis, M.N., Srivastava, D., Aberer, K., Deshpande, A., Florescu, D., Chan, C.Y., Ganti, V., Kanne, C.-C., Klas, W., Neuhold, E.J. (eds.) VLDB, pp. 663–674. ACM, New York (2007)
24. Cafarella, M.J., Halevy, A.Y., Wang, D.Z., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. PVLDB 1(1), 538–549 (2008)
25. Uren, V.S., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. J. Web Sem. 4(1), 14–28 (2006)
26. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley, Reading (2005)

27. King, P.J.H., Poulovassilis, A.: Enhancing database technology to better manage and exploit partially structured data. Technical report bbkcs-00-14, Birkbeck University of London (2000), `http://www.dcs.bbk.ac.uk/research/techreps/2000/bbkcs-00-14.pdf`
28. Bairoch, A., Boeckmann, B., Ferro, S., Gasteiger, E.: Swiss-prot: Juggling between evolution and stability. Briefings in Bioinformatics 5(1), 39–58 (2004)
29. Doan, A., Halevy, A.Y.: Semantic-integration research in the database community. AI Mag. 26, 83–94 (2005)
30. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. Knowl. Eng. Rev. 18, 1–31 (2003)
31. Choi, N., Song, I.-Y., Han, H.: A survey on ontology mapping. SIGMOD Rec. 35, 34–41 (2006)
32. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. 5(3), 1–22 (2009)
33. Talukdar, P.P., Ives, Z.G., Pereira, F.: Automatically incorporating new sources in keyword search-based data integration. In: Elmagarmid, Agrawal [54], pp. 387–398
34. Do, H.H., Rahm, E.: Matching large schemas: Approaches and evaluation. Inf. Syst. 32(6), 857–885 (2007)
35. Talukdar, P.P., Reisinger, J., Pasca, M., Ravichandran, D., Bhagat, R., Pereira, F.: Weakly-supervised acquisition of labeled class instances using graph random walks. In: EMNLP, pp. 582–590. ACL (2008)
36. Dessì, N., Pes, B.: Towards scientific dataspaces. In: Web Intelligence, IAT Workshops, pp. 575–578. IEEE, Los Alamitos (2009)
37. Hamilton, J.: Perspectives: One size does not fit all (2009), `http://perspectives.mvdirona.com/CommentViewguidafe46691-a293-4f9a-8900-5688a597726a.aspx`
38. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: Bressoud, T.C., Frans Kaashoek, M. (eds.) SOSP, pp. 205–220. ACM, New York (2007)
39. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: OSDI [53], pp. 335–350
40. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. In: Scott, M.L., Peterson, L.L. (eds.) SOSP, pp. 29–43. ACM, New York (2003)
41. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI 2004, pp. 137–150 (2004)
42. Dean, J., Ghemawat, S.: Mapreduce: a flexible data processing tool. Commun. ACM 53(1), 72–77 (2010)
43. Dean, J.: Experiences with mapreduce, an abstraction for large-scale computation. In: PACT 2006: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques, p. 1. ACM Press, New York (2006)
44. George, L.: Hbase architecture (2009), `http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html`
45. Apache Foundation Software. Apache hive, data warehouse infrastructure built on top of apache hadoop (2010), `http://hive.apache.org/`
46. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. 2(2), 1626–1629 (2009)

47. Apache Foundation Software. The couchdb project (2008), `http://couchdb.apache.org/`
48. Cloudant.com. Cloudant bigcouch (2008), `https://cloudant.com/`
49. Evans, N.S., GauthierDickey, C., Grothoff, C.: Routing in the dark: Pitch black. In: ACSAC, pp. 305–314. IEEE Computer Society, Los Alamitos (2007)
50. Balakrishnan, H., Frans Kaashoek, M., Karger, D., Morris, R., Stoica, I.: Looking up data in p2p systems. Commun. ACM 46, 43–48 (2003)
51. Yahoo! Searchmonkey (2011), `http://developer.yahoo.com/searchmonkey/`
52. Dayal, U., Whang, K.-Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.-K. (eds.): Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15. ACM, New York (2006)
53. Symposium on Operating Systems Design and Implementation (OSDI 2006), November 6-8. USENIX Association, Seattle (2006)
54. Elmagarmid, A.K., Agrawal, D. (eds.): Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, June 6-10. ACM, USA (2010)